A Unified Framework based on HTN and POP for Approaches for Multi-Agent Planning

Damien Pellier, Humbert Fiorino

A Unified Framework based on HTN and POP Approaches for Multi-Agent Planning

Damien Pellier
Dept. of Mathematics & Computer Science
Paris-Descartes University
45 rue des Saints-Pères F-75270 Paris
Email: Damien.Pellier@math-info.univ-paris5.fr

Humbert Fiorino
Laboratoire d'Informatique de Grenoble
46, avenue Félix Viallet F-38031, Grenoble
Email: Humbert.Fiorino@imag.fr

Abstract

The purpose of this paper is to introduce a multi-agent model for plan synthesis in which the production of a global shared plan is based on a promising unified framework based on HTN and POP approaches. In order to take into account agents' partial knowledge and heterogeneous skills, we propose to consider the global multi-agent planning process as a POP planning procedure where agents exchange proposals and counter-proposal. Each agent's proposal is produced by a relaxed HTN approach that defines partial plans in accordance with the plan space search planning, *i.e.*, plan steps can contain open goals and threats. Agents interactions define a joint investigation that enable them to progressively prune threats, solve open goals and elaborate solutions step by step. This distributed search is sound and complete.

Introduction

The problem of plan synthesis achieved by autonomous agents in order to solve complex and collaborative tasks is still an open challenge. Increasingly new application areas can benefit from this research domain: for instance, cooperative robotics (Alami *et al.* 1998) or composition of semantic web services (Wu *et al.* 2003) when considering actions as services and plans as composition schemes. In this paper, we introduce a multi-agent model for plan synthesis in which the production of a global shared plan is viewed as a collaborative goal directed reasoning about actions. The key idea is to take advantage of two planning approaches: POP planning that is well adapted to produce concurrent plans in distributed environment (because no explicit global state is required) and HTN planning for its expressiveness and performance in order to generate refinements.

At the team's level, each agent can refine, refute or repair the ongoing team plan with a POP procedure (Penberthy & Weld 1992). If the repair of a previously refuted plan succeeds, it becomes more robust but it can still be refuted later. If the repair of the refuted plan fails, agents leave this part of the reasoning and explore another possibility: finally, "bad" sub-plans are ruled out because there is no agent able to push the investigation process further. As in an argumentation with opponents and proponents, the current plan is considered as an acceptable solution when the proposal/counter-proposal cycles end and there is no more objection.

At the agent's level, the specificity of this approach relies on the agent's capabilities to elaborate plans under partial knowledge and/or to produce plans that partially contradict its knowledge. In other words, in order to reach a goal, such an agent is able to provide a plan *which could be executed if certain conditions were met*. Unlike "classical" planners, the planning process does not fail if some conditions are not asserted in the knowledge base, but rather proposes plans where action preconditions are possibly not resolved and are considered as open goals. Obviously, the goal cannot be considered "achieved" and the open goals must be as few as possible because they become new goals for the other agents. For instance, suppose that a door is locked: if the agent seeks to get into the room behind the door and the key is not in the lock, the planning procedure fails even though the agent is able to fulfill 100% of its objectives behind the door. Another possibility is to suppose for the moment that the key is available and then plan how to open the door *etc*. whereas finding the key becomes a new goal to be delegated. To that end, we designed a planner based on the HTN procedure (Nau *et al.* 2003) that relaxes some restrictions regarding the applicability of planning operators.

Our approach differs from former ones in two points. First of all, unlike approaches that emphasize the problem of controlling and coordinating a posteriori local plans of independent agents by using negotiation (Zlotkin & Rosenschein 1990), argumentation (Tambe & Jung 1999), or synchronization (Tonino *et al.* 2002; Clement & Barrett 2003) *etc.*, the unified planning framework presented here focuses on generic mechanisms allowing agents to jointly elaborate a global shared plan and carry out collective actions. Secondly, by elaboration, we mean plan production and not instantiation of predefined global plan skeletons (Grosz & Kraus 1996; D'Inverno *et al.* 2004). This is achieved by composing agents' skills, *i.e.*, the actions they can execute for the benefit of the group. Thus, the issues are: how can agents produce plans as parts of the global team plan with their partial and incomplete beliefs? What kind of refutations, *i.e.,* threats according to POP terminology, and repairs agents can propose to produce robust plans? And how to define such a distributed planning procedure to guarantee its

soundness and completeness?

Plan and Open Goal

In this section, we define the syntax and semantics used in the planning algorithms. In particular, we use the usual first order logic definitions: constant, function, predicate and variable symbols. Propositions are tuples of parameters, (*i.e.*, constants or variables), and can be negated or not. Codesignation is an equivalence relation on variables and constants. Binding constraints enforce codesignation or noncodesignation of parameters and two propositions codesignate if both are negated or both are not negated, and if the tuples are of the same length and if corresponding parameters codesignate. An *operator* is a tuple of the form $o = (\mathit{name}(o), \mathit{precond}(o), \mathit{add}(o), \mathit{del}(o))$: name$(o)$ is denoted by $n(x_1, \ldots, x_k)$ where n is an *operator symbol* and $x_1, \ldots, x_k$ parameters; *precond*(o), *add*(o) and *del*(o) are respectively the preconditions, the operator's add list and delete list. A *method* is a tuple of the form $m = (\mathit{name}(m), \mathit{precond}(m), \mathit{reduction}(m))$: *name*$(m)$ is denoted by $n(x_1, \ldots, x_k)$ where n is the method name and $x_1, \ldots, x_k$ its parameters; *precond*(m) are the method preconditions; *reduction*(m) is a sequence of relevant operators or methods to achieve m. An agent is an autonomous planning process. More formally:

Definition. 1 (Agent) *An* agent *is a tuple of the form* $ag = (\mathit{name}(ag), \mathit{operators}(ag), \mathit{methods}(ag), \mathit{beliefs}(ag))$*:*

- *name*(ag) *identifies the agent;*
- *operators*(ag) *and methods*(ag) *is respectively a set of operators and methods;*
- *beliefs*(ag) *are facts about the world that the agent believes to be* true*.*

It is worth noting that absent facts are not *false* but *unknown*. Moreover, we consider that each agent's beliefs are mutually consistent: if a fact is asserted in one agent's beliefs, it cannot be negated elsewhere.

Definition. 2 (Planning problem) *A* planning problem *is a tuple of the form* $\mathcal{P} = (s_0, \mathcal{T}, g)$*:*

- s_0 *is the union set of the agents' beliefs;*
- $\mathcal{T}$ *is a set of agents;*
- g *is the goal i.e. facts about the world to be achieved by* $\mathcal{T}$*.*

In order to solve a planning problem, the agents seek to find the different steps that will bring them from s_0 to g. These steps are partially ordered actions (concurrency is possible between agents), *i.e.*, operators or methods whose parameters are instantiated according to their corresponding binding constraints. A partial plan is an endeavor to find a solution:

Definition. 3 (Partial Plan) *A* partial plan *is a tuple of the form* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$*:*

- $\mathcal{A} = \{a_0, \ldots, a_n\}$ *is a set of actions;*
- $\prec$ *is a set of ordering constraints on* $\mathcal{A}$ *where* $a_i \prec a_j$ *means* a_i *precedes* a_j*;*
- $\mathcal{I}$ *is a set of binding constraints on action parameters denoted by* $x = y$*,* $x \neq y$*, or* $x = \mathit{cst}$ *such that* $\mathit{cst} \in D_x$ *and* D_x *is the domain of* x*;*
- $\mathcal{C}$ *is a set of causal links denoted by* $a_i \xrightarrow{p} a_j$ *such that* a_i *and* a_j *are two actions of* $\mathcal{A}$*; the ordering constraint* $a_i \prec a_j$ *exists in* $\prec$*; the fact* p *is an effect of* a_i *and a precondition of* a_j*, and the binding constraints about the parameters of* a_i *and* a_j *corresponding to* p *are in* $\mathcal{I}$*.*

When asserting, an agent can use actions whose preconditions are not totally endorsed by causal links. In a multi-agent context, this makes sense because these open goals become sub-goals to be achieved by the other agents.

Definition. 4 (Open Goal) *Let* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ *be a partial plan. An open goal stated by* π *is defined as a precondition* p *of an action* $a_j \in \mathcal{A}$ *such that, for all actions* $a_i \in \mathcal{A}$*, the causal link* $a_i \xrightarrow{p} a_j \notin \mathcal{C}$*. opengoals*$(\pi)$ *and opengoals*(a_j) *respectively denote the set of open goals stated by* π *and* a_j*.*

We assume that $\prec$ is consistent: it is possible to find at least one total order compliant with $\prec$, (*i.e.*, a completion). In other terms, there is no cycle in $\mathcal{A}$ and $\prec$ represents a class of total orders.

Definition. 5 (Linearization) *Let* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ *be a partial plan.*

1. *A linearization of* π *is a partial plan* $\lambda = (\mathcal{A}, <, \mathcal{I}, \mathcal{C})$*, where* $<$ *is a total order on* $\mathcal{A}$ *consistent with* $\prec$*, that defines a sequence of* $n+1$ *states* $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ *with*

   $s_i = (((s_{i-1} \cup \mathit{opengoals}(a_{i-1})) - \mathit{del}(a_{i-1})) \cup \mathit{add}(a_{i-1})$

   for $0 \leq i \leq n$*.*
2. *A completion of* π *is the set of all the linearizations of* π *denoted by completion(*π*).*

Solution plan and refutation

Obviously, planning is finished when the completion of a partial plan solves the assigned goal.

Definition. 6 (Solution plan) *A partial plan* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ *is a solution plan for* $\mathcal{P} = (s_0, \mathcal{T}, g)$ *if:*

- *the sets of ordering constraints* $\prec$ *and of binding constraints* $\mathcal{I}$ *are consistent;*
- *all the linearizations* $\lambda \in$ *completion(*π*) define a sequence of states* $\langle s_0, \ldots, s_i, \ldots s_n \rangle$ *for* $0 \leq i \leq n$ *such that:*
  - *the goal* g *is satisfied in* s_n*, i.e.,* $g \subseteq s_n$*;*
  - λ *does not state open goals, i.e.,* $\mathit{opengoals}(\lambda) = \emptyset$*.*

But the number of linearizations of a partial plan is exponential in its size and computing them in order to decide whether or not a partial plan is a solution plan would be time consuming. Therefore, we propose a necessary criterion based on the notion of refutation to fix the correctness of a partial plan (Chapman 1987).

Definition. 7 (Refutation) *A refutation on* $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ *is a tuple* $(a_k, a_i \xrightarrow{p} a_j)$ *such that:*

- *a_k produces the effect $\neg q$, and, p and q codesignate;*
- *the ordering constraints $a_i \prec a_k$ and $a_k \prec a_j$ are consistent with $\prec$;*
- *the binding constraints corresponding to the codesignation of p and q are consistent with $\mathcal{I}$.*

Proposition. 1 *A partial plan $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ is a solution plan for a problem $\mathcal{P}$ if:*

- *$\prec$ and $\mathcal{I}$ are consistent;*
- *π contains no threat, i.e., neither open goal nor refutation.*

Here is a lemma necessary to prove the proposition 1:

Lemma. 1 *Let $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ be a partial plan without open goal and $(a_i \xrightarrow{p} a_n) \in \mathcal{C}$ a causal link. Necessarily $p \in s_n$ if there is no refutation $(a_k, a_i \xrightarrow{p} a_n)$.*

Proof. 1 *This lemma is proved by induction on the length of $\lambda \in$ completion(π):*

Base step: let $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ with $\mathcal{A} = \{a_0, a_\infty\}$. completion$(\pi) = \{\lambda\}$ and $\lambda = \langle a_0, a_\infty \rangle$. $s_0 = s_n$ and, by definition, there is no possible refutation ($\forall p \in s_0, p \in s_n$).

Induction step: suppose that the lemma is true for π with n actions. Let us prove this is also true for π with $n+1$ actions. Let $\lambda \in$ completion(π) with $\lambda = \langle a_0, \ldots, a_{n-1}, a_n \rangle$ ($a_n = a_\infty$) and $\lambda' = \langle a_0, \ldots, a_{n-1} \rangle$. By induction, $\forall (a_i \xrightarrow{p} a_{n-1})$ for $0 \leq i < n-1$, $p \in s_{n-1}$ if there is no refutation $(a_k, a_i \xrightarrow{p} a_{n-1})$ for $0 \leq k < n-1$. By definition, $s_n = (s_{n-1} - del(a_{n-1})) \cup add(a_{n-1})$. Thus, $\forall p \in s_n$, either $p \in add(a_{n-1})$, or $p \in s_{n-1}$ and $p \notin del(a_{n-1})$. In the former case, p is produced by a_{n-1} and there is no possible refutation. In the latter case, p is produced by λ' and it is not refuted by a_{n-1}. Thus, in any case, the lemma is true.

Proof. 2 *Proposition 1 proof is as follows: let $\pi = (\mathcal{A}, \prec, \mathcal{I}, \mathcal{C})$ be a partial plan. $\prec$ and $\mathcal{I}$ are consistent and there is no threat in π. Therefore, $\forall \lambda \in$ completion(π), λ defines a sequence of states $\langle s_0, \ldots, s_n \rangle$ such that $s_i = (s_{i-1} - del(a_{i-1})) \cup add(a_{i-1})$ because opengoals$(\pi) = \emptyset$. As there is no refutation in π, $g \subseteq s_n$. This can be proved by contradiction: suppose it exists $p \in g$ and $p \notin s_n$. As p cannot be an open goal ($\exists (a_i \xrightarrow{p} a_n)$), the absence of p in s_n is due to a refutation according to the lemma 1. This is contradictory with the absence of threat. Thus, $g \subseteq s_n$ and π is a solution plan.*

Team Planning Procedure

Agents Interactions Rules. The agents involved in the process of cooperatively building a solution plan are committed to follow several interactions rules so as to guarantee the validity of the reasoning as well as the correctness of the proposed solution plan. Each agent handles its own view of the space search in which it records the propositions of the other agents. The space search is a Directed Acyclic Graph (DAG) whose nodes are partial plans and edges are *refinement*, *refutation* or *repair* operators. The agents produce two kinds of interactions: *informational interactions* such as *refine*, *refute*, *repair*, *failure*, that enable them to exchange information about the partial plans contained in their space search, and *contextualization interactions* such as *prop.solve*, *prop.failure*, *prop.success*, *ack.failure*, *ack.success*, that enable them to set the interaction context. Each agent's search space is initiated with π_0: $\mathcal{A} = \{a_0, a_\infty\}$ such that *precond*$(a_\infty) = g$ and *add*$(a_0) = s_0$; $(a_0 \prec a_\infty)$; the binding constraints corresponding to a_0, a_∞ and an empty set of causal links. The agents broadcast asynchronous messages whose delivery time is finite. All messages are delivered in the causal order of their emission so as to ensure that their space search remains consistent with each other. The informational interactions comply with the agent's rationality and update rules given in table 1.

Contextualization Rules. This section explains how the agents start and stop the plan synthesis. The corresponding contextualization rules are represented as a finite state automaton (*cf.* figure 1) whose states are the dialog states and edges are the dialog acts (*cf.* table 1). ? and ! before an act respectively means that the act is received or sent by the agent.

Initially, the agents are in **IDLE** state: they wait for a goal to solve. When sending or receiving *prop.solve*, they switch to **Planning** state in which they exchange refinements, refutations and repairs according to table 1. There are two possibilities to leave this state: either the agent proposes to stop or it receives a proposition to stop. In the former case, the agent releases the act *prop.failure* (resp. *prop.success*) to stop with failure (resp. to stop with success). Then, the automaton switches to the **Failure** state (resp. **Success**) in which the agent waits the local acknowledgments of the other agents. If all teammates acknowledge the proposition, the stopping conditions with failure (resp. with success) are satisfied. In the latter case (when the agent is requested to stop), the interactions closing needs a two steps procedure.

The first step is the verification one: the interactions switche either to the **IF** state, *i.e.*, Instance Failure or to the **IS** state i.e. Instance Success. In the case of a stopping proposition with failure, the agent must verify if it cannot provide a solution and, in case of a stopping proposition with success, it must verify that the proposed plan is a solution plan. Indeed, it can still refute it on the grounds of facts ignored by the proposer. For instance, effects that do not support preconditions through causal links are not communicated and can generate refutations.

The second step is the acknowledgment one: if stopping with failure (resp. with success) is asserted locally, the agent acknowledges the stopping proposition by releasing *ack.failure* (resp. *ack.success*) and the interations changes to the **Failure** state (resp. **Success**). The **Failure** and **Success** states correspond to an agent's waiting of the other agents' acknowledgments. It is worth noting that the non acknowledgment of a stopping proposition conveys no specific contextualization act. Indeed, suppose an agent is in **IF**: if it receives or sends a refinement, a refutation or a repair, that means at least an agent is able to carry on the planning pro-

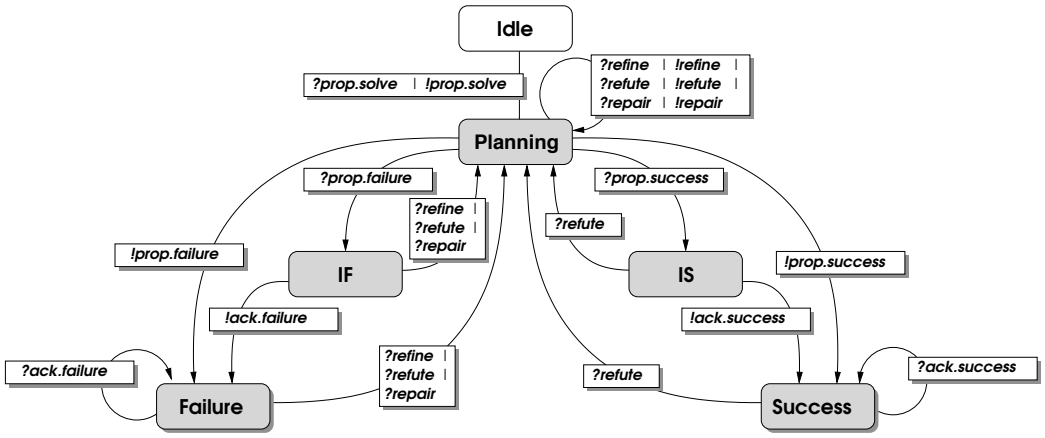

Figure 1: Contextualization automaton

cess. Therefore, the agents must reassess their space search because it has been modified. Similarly, if an agent, which is in **IS**, receives or sends a refutation, the proposed plan is rejected and the planning process must be continued. In any case, the agents return to **Planning**.

Agent Planning Algorithm. The agent planning algorithm is given by the algorithm 1. First of all, a non terminal partial plan is nondeterministically chosen within the search space (a partial plan is terminal if no refinement, repair or refutation is applicable). If all the partial plans are terminal (line 3), investigation cannot be pushed further and the agent proposes to stop with failure and ends its reasoning loop (line 5). The stopping proposition is submitted to its teammates.

If at least one of the partial plan is non terminal, the agent must assess whether or not this partial plan is a solution plan: in this case, the plan π does not contain any open goal and cannot be refuted. The former condition is verified by the procedure `OpenGoals(`π`)`. This procedure can be efficiently implemented with a hash table: each time a new action is added, its open goals are added to the table, and, whenever a causal link is added, the corresponding open goal is withdrawn. The latter condition is checked by the procedure `Threat(`π`)`.

When a plan is considered to be a solution plan (line 9), the agent proposes to stop the investigation process and ends its reasoning loop. Otherwise, it proposes to apply a refinement, a repair or a refutation. Let suppose it decides to apply a refinement (line 14): `Refine(`ϕ, π`)` returns all the possible refinements where ϕ are the open goals of π. If there is no possible refinement, ϕ is labeled as unsolved and the agent broadcasts the failure. Otherwise, the agent picks up a refinement and submits it to the other agents. The same mechanism is applied for repairs (line 24). Finally, if ϕ is a refutation (line 35), the agent broadcasts it to the other agents. The main characteristic of the agent planning procedure is the following:

Proposition. 2 *The multi-agent planning procedure is sound and complete: every time there is a solution for $\mathcal{P} = (s_0, \mathcal{T}, g)$ from an agent's view, this agent ends its procedure by releasing a proposition of success and a solution plan.*

Proof. 3 (Soundness) *By definition, the initial partial plan π_0 is defined by $\mathcal{A} = \{a_0, a_\infty\}$, where $\prec$ and $\mathcal{I}$ are respectively a consistent set of ordering constraints and binding constraints. When an agent makes a proposition of success, the proposed plan π contains no more threat and all the refinement and repair operators keep $\prec$ and $\mathcal{I}$ consistent. This is guaranteed by the interactions rules. Thus, according to proposition 1, π is a solution plan.*

But the multi-agent planning procedure is also sound from the group's view because the interactions' cycle ends if and only if an agent has released a proposition of success about π (π locally satisfies the properties of a solution plan) and π is not refuted by the other agents. Thus, π is a solution plan for $\mathcal{P}$ at the group's level.

Proof. 4 (Completeness) *We must prove that, at least, one execution trace of the multi-agent planning procedure returns a solution plan from an agent's view whenever it exists. The proof is by induction on the solution plan length k.*

Base step ($k = 0$): the empty plan π is solution of $\mathcal{P}$. Thus, π contains no threat and the multi-agent procedure immediately releases a success proposition.

Induction step: let us suppose that the multi-agent procedure is complete for problems having k-length solutions. Let $\mathcal{P} = (s_0, \mathcal{T}, g)$ having a $(k+1)$-length solution plan $\langle a_0, \ldots, a_k \rangle$. As a_k is relevant to achieve the goal g, there is at least one agent that nondeterministically will propose an operator to apply to the initial plan π_0 and adding a_k. Let $s_{k+1} = ((s_k \cup opengoals(a_k)) - del(a_k)) \cup add(a_k)$, the state following the application of a_k. At the next recursive call of the procedure, the plan π_1 is, at least, composed of tree actions $\{a_0, a_k, a_\infty\}$. π_1 is similar to the initial plan of a problem defined by the state s_0 and $g = opengoals(a_k) \cup opengoals(a_\infty)$. This problem admits a solution $\langle a_0, \ldots, a_j \rangle$ with a length $j \leq k$. By induction, the recursive calls of the procedure from π_1 build up a trace that finds a solution plan $\langle a_0, \ldots, a_j \rangle$ and the algorithm proposes to stop with success.

Multi-agent Planning Heuristics. As the search space is a DAG, it can be explored with A*. But this raises many problems.First of all, A* is not a distributed algorithm and

refine(ρ, p, π) : ρ is a refinement and p the refined open goal of partial plan π.

Rationality: the agent ensures that:

- π is a partial plan of its space search;
- h is an open goal of π;
- ρ has not been proposed as refinement of p;
- the partial plan π', result of ρ, contains consistent binding and ordering constraints.

Dialog: the agents can either refine all the open goals of π' or refute π'.

Update: add π' as refinement of p in π into its space search.

refute(ϕ, π) : ϕ refutes π.

Rationality: the agent ensures that:

- π is in its space search;
- ϕ has not been proposed as refutation of π.

Rules: the agents can repair π.

Update: add ϕ as refutation of π into its space search.

repair(ψ, ϕ, π) : ψ is a repair of π corresponding to the refutation ϕ.

Rationality: the agent ensures that:

- π is in its space search;
- ϕ refutes π;
- ψ has not been proposed as repair of π about the refutation ϕ;
- the partial plan π' provided by ψ contains consistent binding and ordering constraints.

Rules: either its teammates refine all the open goals of π' or they refute π'.

Update: add π' as repair of the refutation ϕ in π.

failure(Φ,π) :

Rationality: the agent ensures that:

- π is in its space search;
- Φ is a threat of π;

Rules: $\emptyset$

Update: label Φ as unsolved by the agent that produces the interaction.

Table 1: Interactions Rules

agents' interactions are asynchronous. It would have been possible to implement a deliberation protocol in order to jointly choose among the pending plans which one is the more promising. But this approach has some drawbacks. Deliberations would be time consuming and would not take advantage of possible concurrent explorations; agents unable to cooperate at a moment to the planning procedure would be idle until another plan is chosen. For those reasons, we propose that all the agents implement the same heuristic function.

Then, the issue is how to tailor an appropriate heuristic function $f(\pi) = g(\pi) + h(\pi)$? To find an optimal solution

Algorithm 1: Agent planning procedure

```
1  while planning = true do
2      plans ← non terminal plan in the space search;
3      if plans is empty then
4          Submit failure proposition;
5          planning ← false ;
6      else
7          Select a plan π ∈ plans ;
8          flaws ← OpenGoals(π) ∪ Threats(π);
9          if flaws is empty then
10             Submit a success proposition about π;
11             planning ← false ;
12         else
13             Select a threat φ ∈ flaws ;
14             if φ is an open goal then
15                 refinements ← Refine(φ, π);
16                 if refinements is empty then
17                     Label φ as unsolved;
18                     Assert failed repair of φ;
19                 else
20                     Select refinement ρ ∈ refinements ;
21                     Submit ρ as refinement φ of π;
22             else if φ is a threat already asserted then
23                 repairs ← Repair(φ, π);
24                 if repairs is empty then
25                     Label φ as impossible to solve;
26                     Assert the failed repair of φ;
27                 else
28                     Select repair ψ ∈ repairs ;
29                     Submit ψ as repair of φ in π;
30             else
31                 Submit φ as new refutation of π;
```

plan, $h(\pi)$ must be admissible and never overestimate the distance to a solution[1]. To that end, $g(\pi)$ is the number of actions and $h(\pi)$ is the number of open goals. Alternatively, $g(\pi)$ could be the number of causal links. Indeed, each action is at least connected to another one through a causal link. But plans that have several open goals refined by an action would be penalized, which is not convenient.

Moreover, when a plan is chosen, another decision must been taken: should the plan be refuted, refined or repaired? This question can expressed itself how an agent must select a flaw, *i.e.*, an open goal or a threat. Obviously, all open goals and refutations must be fixed by a resolver refining or repairing the current plan. But, when several refinements or repairs are possible, only one of them is required. Therefore, an agent must choose the flaw having the smallest number

[1] As shown by (Penberthy & Weld 1992; Gerevini & Schubert 1996), $h(\pi)$ is only admissible as long as the refinement cost from one partial plan to the next is not null. Indeed, $h(\pi)$ sometimes overestimates the distance to a solution because an open goal refinement not always entails the addition of actions.

of resolvers. The rational for that is to work on flaws with the smallest branching factor as early as possible in order to limit the cost of eventual space search backtrack. The FAF heuristic is easy to compute in $\mathcal{O}(n)$, where n is the number of flaws in a plan. Furthermore, experimental results (Pollack, Joslin, & Paolucci 1997) have shown that FAF works relatively well compared to other flaw selection strategies.

Finally, when many resolvers are available to solve a flaw, an agent must decide which resolver is the best candidate node at the search point. Let $\pi_1, \ldots, \pi_n$ be the set of open nodes at some search point and *opengoals*(π) the set of propositions in π without causal link. A very simple and intuitive heuristic is to choose a partial plan π that has the smallest number of open goals. We will introduce how this heuristic is implemented in the last section. Note, there are others resolvers selection heuristics (Ghallab & Laruelle 1994), which are based on the initial state of the planning problem. However, these heuristics are more difficult to implement due to the distribution of the initial state among the agents.

Refinement mechanisms

In this section, we present the different refinement mechanisms.

Causal link addition. This is the simplest refinement: if p is an open goal of a_j and there is an action a_i that produces p, then $(a_i \xrightarrow{p} a_j)$ is added to $\mathcal{C}$. Causal link addition allows to take advantage of the favorable relations between actions in a plan. Moreover, it enables the agents to iteratively build a_0 (i.e. s_0). This initial state is scattered over the agents, but thank to causal link additions relevant initial facts are broadcasted. More generally, this kind of refinement is important because it provides a rational means to share beliefs.

Sub-plan addition. In this kind of refinement, a sub-plan is added to support an open goal. But, this sub-plan can itself contain open goals that will have to be solved *etc*. This can be achieved by our relaxed HTN planning approach, which is detailed later.

Refutation mechanism. The refutations have been defined in definition 7: computing a refutation means to find an action a_k that invalidates a causal link $a_i \xrightarrow{p} a_j$.

Repair mechanisms

Repair mechanisms compute the modifications to be done on refuted plan. We distinguish ordering constraints addition, binding constraints modification and sub-plan addition.

Ordering constraints addition. In this kind of repair, ordering constraints are added in order to prevent the rebutting action a_k from being executed between a_i and a_j. The different possibilities are listed in table 2. It is worth noting that, when the plan cannot be repaired by ordering constraints addition, it can eventually be repaired by adding a sub-plan.

Sub-plan addition. When a_k deletes p necessary to a_j, sub-plan addition must enforce the production of p after a_k. This can be achieved by our relaxed HTN planning approach, which is detailed in the next section.

Existing constraints on a_k	**Constraints to add**
$\emptyset$	$a_j \prec a_k$ or $a_k \prec a_i$
$a_k \prec a_j$	$a_k \prec a_i$
$a_i \prec a_k$	$a_j \prec a_k$
$a_i \prec a_k$ and $a_k \prec a_j$	no solution

Table 2: Ordering constraints addition

Binding constraints modification. Another way to repair a refutation is to prevent the undercutting facts p and $\neg q$ from codesignating. To that end, binding constraints can be modified as long as consistency is preserved.

Refinement based on HTN Planning

Relaxed HTN and refinement. We postulate that operators or methods can be triggered even though some preconditions are not satisfied by the agent's beliefs. These preconditions are *assumed* by the agent. Formally, open goal computation is based on the identification of all the possible substitutions to apply a method or an operator. The results of these substitutions are respectively *complex* or *primitive* actions. Furthermore, there are two kinds of open goals: *hypotheses* are facts not included in the agent's beliefs; *denials* are facts that correspond to negated beliefs. In the former case, the agent requests its teammates to provide a resource. In the latter case, the agent believes p but submits $\neg p$ as a sub-goal for its teammates.

The refinement algorithm is based on (Nau *et al.* 2003). We consider that the goal g of a planning problem represents the preconditions of a complex action, *i.e.*, a sequence of primitive or complex actions. Then refinement is defined as follows:

Definition. 8 (Refinement) *Let $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$ be a planning problem. Plan π is a refinement of $\mathcal{P}$ if all the linearizations $\lambda \in$ completion(π) refine $\mathcal{P}$. Let $\lambda = \langle a_0, \ldots, a_k \rangle$ be a sequence of primitive actions, λ refines $\mathcal{P}$ if one of the following conditions is true:*

Case 1: the sequence of actions to be achieved is empty then λ is empty;

Case 2: α_0 is primitive: α_0 is relevant for a_0, α_0 is applicable from s_0 and $\lambda = \langle a_1, \ldots, a_k \rangle$ refines $(s_1, \mathcal{T}, \langle \alpha_1, \ldots, \alpha_n \rangle)$;

Case 3: α_0 is complex: there is a reduction $\langle r_1, \ldots, r_j \rangle$ applicable to achieve α_0 from s_0 and λ refines $(s'_0, \mathcal{T}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$ with $s'_0 = s_0 \cup$ opengoals(α_0).

Algorithm 2 implements the definition 8. It explores a state space called refinement tree. Each node of the tree is a tuple $(s, \langle \alpha_0, \ldots, \alpha_n \rangle)$ where s is the state of the world and $\langle \alpha_0, \ldots, \alpha_n \rangle$ the remaining actions at this decomposition step. The edges are the possible transition between the different states. Each transition is labeled by the applied operator or method with its binding constraints and its possible open goals.

The refinements are produced in two steps: the expansion of the refinement tree which is detailed below and the plan

extraction which is built on a path from the root to a solution leaf. The refinement tree construction is based on the recursive decomposition of the complex actions contained in the nodes until a leaf is produced i.e. a node with an empty sequence of actions. This leaf represents the state reached after the refinement execution. The expansion procedure begins with the agent's initial beliefs s_0 and the sequence of the actions to realize $\langle \alpha_0, \ldots, \alpha_n \rangle$. First of all, the procedure tests if this sequence is empty (line 2). In this case, a leaf is reached and the initial node is returned as solution. Otherwise, the procedure tries to achieve the first action α_0. Two cases must be considered according to α_0:

1. α_0 **is primitive.** For each operator of $\mathcal{O}$, the procedure tests if it can achieve α_0 (line 3). In this case, the procedure computes the substitutions codesignating the operator preconditions with the current state s contained in n (line 7). Finally, for each substitution σ, the algorithm adds a successor to n (*cf.* case 2 of definition 8).
2. α_0 **is complex.** The procedure is based on the same principle. However, this time, the procedure tests the methods enable to achieve a_0 and applicable in s (line 17). For each of this method, a successor is added to n (*cf.* case 3 of definition 8).

Finally, a node n_c is nondeterministically chosen among the successors of n (line 32) and the procedure is called recursively with this node as parameter.

Proposition. 3 (Soundness) *Let $\lambda = \langle a_0, \ldots, a_k \rangle$ be a nondeterministic execution trace. If* `Refine`$(n, \mathcal{O}, \mathcal{M})$ *returns a node $(s_k, \langle\rangle)$ then λ refines $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.*

Proof. 5 *The proposition is proved by induction on m, the number of calls to the procedure* `Refine`*:*

Base step ($m = 1$): there is only one call to `Refine`*. Therefore, the procedure ends at line 2 with the node $(s_0, \langle\rangle)$. The returned node is the root node, $\lambda = \langle\rangle$ and λ refines $\mathcal{P} = (s_0, \mathcal{T}, \langle\rangle)$.*

Induction step: let $m > 1$. Suppose that the proposition is true for $m < m'$. Two cases must be considered:

Case 1: α_0 is primitive. Let α_0 be the first action of $n = (s_0, \langle \alpha_0, \ldots, \alpha_n \rangle)$ and $\lambda = \langle a_1, \ldots, a_k \rangle$ the trace provided by the procedure at line 33 when called with the node $n_c = (s_1, \langle \alpha_1, \ldots, \alpha_n \rangle)$ and $s_1 = ((s_0 \cup \mathit{opengoals}(\alpha_0)) - \mathit{del}(\alpha_0)) \cup \mathit{add}(\alpha_0)$. By induction, λ refines $\mathcal{P} = (s_1, \mathcal{T}, \langle \alpha_1, \ldots, \alpha_n \rangle)$. But α_0 is primitive and achieves a_0 from s_0. Thus, as stated by case 2 of definition 8, $\lambda = \langle a_0, \ldots, a_k \rangle$ refines $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.

Case 2: α_0 is complex. Let α_0 be the first action of $n = (s_0, \langle \alpha_0, \ldots, \alpha_n \rangle)$ and λ the trace provided by the procedure at line 33. The chosen node to extend the tree is $n_c = (s'_0, \langle \mathit{reduction}(\alpha_0), \alpha_1, \ldots, \alpha_k \rangle)$ with $s'_0 = s_0 \cup \mathit{opengoals}(\alpha_0)$ and $\mathit{reduction}(\alpha_0) = \langle r_1, \ldots, r_j \rangle$ (line 25). By induction, λ refines $\mathcal{P} = (s'_0, \mathcal{T}, \langle r_1, \ldots, r_j, \alpha_1, \ldots, \alpha_n \rangle)$. But, α_0 is complex and the reduction achieves a_0 from s_0. Thus, as stated by case 3 of definition 8, $\lambda = \langle a_0, \ldots, a_k \rangle$ refines $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$.

Algorithm 2: Refine(n, $\mathcal{O}$, $\mathcal{M}$)

```
1  Let n = (s, ⟨α_0, ..., α_n⟩) ;
2  if ⟨α_0, ..., α_n⟩ is an empty sequence then return n ;
3  else if α_0 is primitive then
4      foreach operator o ∈ O do
5          θ ← Unify(name(α_0), name(o)) ;
6          if θ ≠ Failure then
7              Σ ← Satisfy(precond(o), s, θ) ;
8              if Σ is empty then return Failure ;
9              foreach substitution σ ∈ Σ do
10                 G ← OpenGoals(precond(o), σ, s) ;
11                 add a successor to n with a_0 = σ(o)
                   such that (((s ∪ G) − del(a_0)) ∪
                   add(a_0)), ⟨α_1, ..., α_n⟩) ;
12             else
13                 return Failure ;
14 else if α_0 is complex then
15     foreach method m ∈ M do
16         θ ← Unify(name(α_0), name(m)) ;
17         if θ ≠ Failure then
18             Σ ← Satisfy(precond(m), s, θ) ;
19             if Σ is empty then return Failure ;
20             foreach substitution σ ∈ Σ do
21                 G ← OpenGoals(precond(m), σ, s) ;
22                 add a successor to n with a_0 = σ(m)
                   such that ((s ∪ G),
23                 Append(reduction(a_0), ⟨α_1, ..., α_n⟩)) );
24             else
25                 return Failure ;
26 nondeterministically choice a node n_c ∈ successor(n);
27 Refine(n_c, O, M) ;
```

Proposition. 4 (Completeness) *Let $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$ be a problem for which a refinement exists. There is at least one nondeterministic trace $\lambda = \langle a_0, \ldots, a_k \rangle$ of* `Refine`$(n, \mathcal{O}, \mathcal{M})$ *that returns a node $(s_k, \langle\rangle)$ such that λ refines $\mathcal{P}$.*

Proof. 6 (Completeness) *The procedure* `Refine` *explores a tree that has a finite branching factor. Indeed, there is a limited number of operators and methods in $\mathcal{P}$ applicable to achieve a specific action. This limitation is due to the fact that a_0 is triggered to achieve α_0 only if name(a_0) can be unified with name(α_0). Moreover, the nondeterministic choice of a pending node (line 33) ensures that, if there is a finite path, it will be found. Therefore,* `Refine` *is complete. More formally, the completeness can be proved by induction on the length of the refinement λ.*

Base step ($m = 0$): $\mathcal{P} = (s_0, \mathcal{T}, \langle\rangle)$ admits the refinement $\lambda = \langle\rangle$. This refinement is immediately returned at line 2.

Induction step: suppose the completeness for refinements with a length inferior to m. Let $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_0, \ldots, \alpha_n \rangle)$. Two cases must be considered:

Case 1: α_0 is primitive. The only way that $\mathcal{P}$ has a refinement with a length m is that a refinement with a length $j < m$ is returned at line 33 for $\mathcal{P} = (s_0, \mathcal{T}, \langle \alpha_1, \ldots, \alpha_n \rangle)$. By induction, this recursive call finds it and, thus, the initial call returns a solution with a length m by adding a_0 on the left of the trace returned at line 33. Indeed, the refinement of $\langle \alpha_1, \ldots, \alpha_n \rangle$ is necessarily smaller than the one of $\langle \alpha_0, \ldots, \alpha_n \rangle$.

Case 2: *α_0 is complex. The proof is similar to case 1.*

Of course, the deterministic implementation of the algorithm does not necessarily guarantee the completeness. This problem can be solved by constraining the description of the methods so as to limit the number of their recursive calls. Another solution is to apply a breadth-first search, which is time consuming. Moreover, the agent must produce reasonable and robust plans. That means plans must contain as few open goals as possible in order to facilitate their refinement as explained in the heuristics section. In the same way, plans are easier to refute, *i.e.*, less robust if they have lots of causal links. Thus, a kind of greedy search is convenient: at each recursion, the chosen node in the refinement tree is the one containing the less open goals within the list of the pending nodes. In case of equality, decision is made according to the number of causal links. An exploration bound is defined as the number of open goals that can be done in a refinement: this limits the refinement tree expansion to refinements whose number of open goals is inferior to this bound and enables to adapt the search to the computing capabilities of each agent.

Conclusion

In this paper, we introduced a multi-agent model for plan synthesis in which the production of a global shared plan is viewed as a collaborative goal directed reasoning about actions. At the team's level, each agent can refine, refute or repair the ongoing team plan based on the POP procedure. At the agent's level, agents elaborate plans under partial knowledge and/or to produce plans that partially contradict its knowledge in order to propose refinement. To that end, we designed a planner based on HTN procedure that relaxes some restrictions regarding the applicability of planning operators.

The advantages of the presented model is to enable agents to compose dynamically their heterogeneous skills and beliefs through a unified planning framework based on POP and HTN procedures. Moreover, it structures the interactions as a collaborative sound and complete investigation process and former works on synchronization, coordination and conflict resolution are integrated through the notions of refutation/repair. From our point of view, this approach is suitable for applications in which agents share a common goal and in which the splitting of the planning and the coordination steps becomes difficult due to the agents strong interdependence: usually, when agents have independent goals, they classically generate plans and then solve their conflicts with specific multi-agent protocols.

The line of work presented here opens two areas of future research. First, we are interested in to testing classical POP heuritics on our framework to study in a distributed manner which are more appropriate, *e.g.*, in terms of the number of exchanged messages. Secondly, we want extend our planning framework for continual planning and generalize the concept of open goals and refutation to properties which are discovered during plan execution.

References

Alami, R.; Fleury, S.; Herrb, M.; Ingrand, F.; and Robert, F. 1998. Multi robot cooperation in the martha project. *IEEE Robotics and Automation Magazine* 5(1):36–47.

Chapman, D. 1987. Planning for conjunctive goals. *Artificial Intelligence* 32(3):333–377.

Clement, B., and Barrett, A. 2003. Continual coordination through shared activities. In *Proceedings of the International Conference on Autonomous Agent and Muti-Agent Systems*, 57–67.

D'Inverno, M.; Luck, M.; Georgeff, M.; Kinny, D.; and Wooldridge, M. 2004. The dmars architecture: A specification of the distributed multi-agent reasoning system. *Autonomous Agents and Multi-Agent Systems* 9(1-2):5–53.

Gerevini, A., and Schubert, L. 1996. Accelerating partial-order planners: Some techniques for effective search control and pruning. *Journal of Artificial Intelligence Research* 5(1):95–137.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the Artificial Intelligence Planning Systems*, 61–67.

Grosz, B., and Kraus, S. 1996. Collaborative plans for complex group action. *Artificial Intelligence* 86(2):269–357.

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, Y. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20(1):379–404.

Penberthy, J., and Weld, D. 1992. Ucpo: A sound, complete, partial order planner for ADL. In B. Nebel, C. R., and Swartout, W., eds., *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 103–114. Morgan Kaufmann Publishers.

Pollack, M.; Joslin, D.; and Paolucci, M. 1997. Flaw selection strategies for partial-order planning. *Journal of Artificial Intelligence Research* 6(1):223–262.

Tambe, M., and Jung, H. 1999. The benefits of arguing in a team. *Artificial Intelligence Magazine* 20(4):85–92.

Tonino, H.; Bos, A.; de Weerdt, M.; and Witteveen, C. 2002. Plan coordination by revision in collective agent-based systems. *Artificial Intelligence* 142(2):121–145.

Wu, D.; Parsia, B.; Sirin E, J.; and Nau, D. 2003. Automating daml-s web services composition using shop2. In *Proceedings of International Semantic Web Conference*.

Zlotkin, G., and Rosenschein, J. 1990. Negotiation and conflict resolution in non-cooperative domains. In *Proceedings of the American National Conference on Artificial Intelligence*, 100–105.